Auf der Suche nach der verlorenen Kataloganreicherungen

Tobias Rademacher

Universitätsbibliothek Leipzig

27. Juni 2012

Gliederung

Kataloganreicherungen

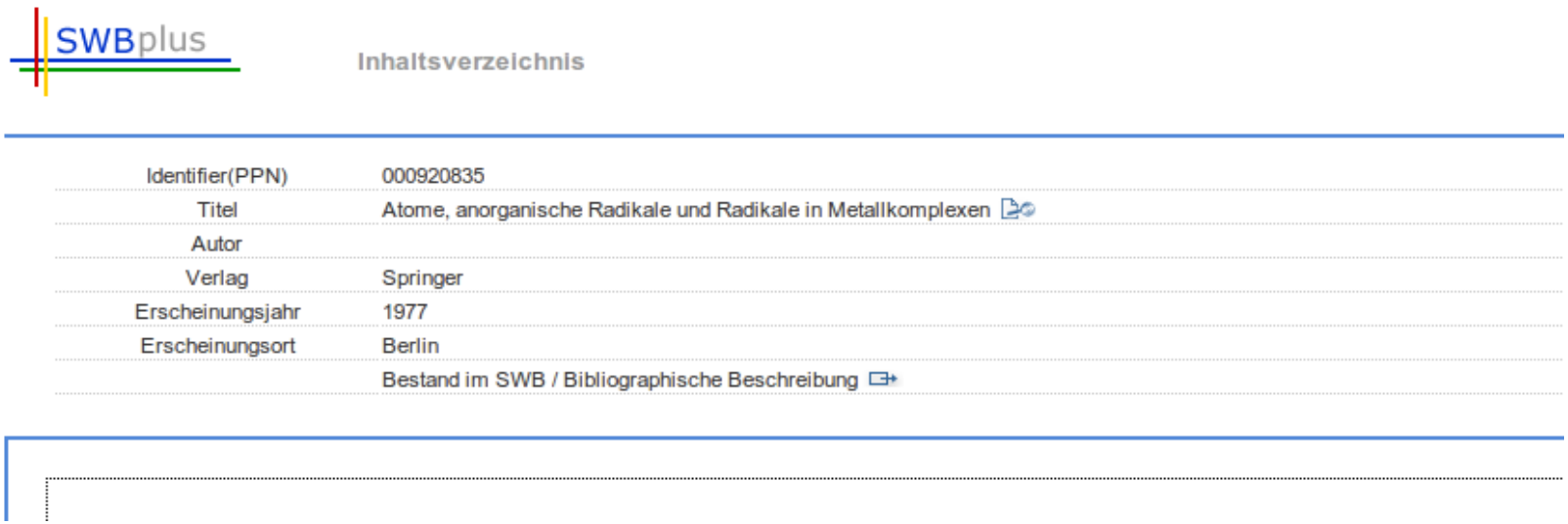
- Inhaltsverzeichnissen
- Abstracts
- Rezensionen
- etc.

SWBplus

- Südwestdeutscher Bibliotheksverbund
- Verknüpfung von Kataloganreicherungen mit Katalogen der Verbundbibliotheken
- Digitalisierung von »Inhaltsverzeichnisse[n], Klappentexte[n] und Rezensionen«[1] (Scans)
- Ziel: qualitativ ›vollwertigere‹ Recherchemöglichkeiten anbieten können
- verbundübergreifender Austausch von digitalisierten (Scans) Beständen von Inhaltsvezeichnissen
- Kooperation mit Verlagen: »Verlagsinformationen, Cover, Leseproben etc.«[2]

[1] *SWBplus* 2012
[2] *SWBplus* 2012

SWBplus Website



Apache Solr

Features I

- »open source enterprise search platform«[3]
- Volltextsuche
- Umfassende Abfragemöglichkeiten
- offene Schnittstellen
- Faceted Search and Filtering

[3] *Apache Solr* 2012

Apache Solr

Features II

- Konfigurierbare Textanalyse (↔ Indexierung)
- Rich Document Parsing → Apache *Tika*
- offene Schnittstellen
- Management Sever/Monitoring (JMX)
- intern: *Lucene*-Engine

Schema

Solr Schema für Kataloganreicherungen

- ›straightforward‹
- *Fields* entsprechend Anforderungen
- Deklaration von »Copy Fields«

Schema

Solr Schema für Kataloganreicherungen

```
1 <field name="id" type="string" indexed="true"  stored="true" required="true"/>
2 <field name="ppn" type="string"  indexed="true" stored="true" required="true"/>
3 <field name="abstract" type="text_general" indexed="true" stored="true"/>
4 <field name="tableofcontents" type="text_general" indexed="true" stored="true"/>
5
6 <uniqueKey>id</uniqueKey>
7
8 <copyField source="abstract" dest="text"/>
9 <copyField source="tableofcontents" dest="text"/>
```

ContentStreaming

ExtractingRequestHandler

- Rich Document Parsing Apache *Tika* (Apache *PDFBox*)
- SolrJ `ContentStreamUpdateRequest` (Java-Klasse)

ContentStreaming

ExtractingRequestHandler

```
1
2   SolrServer server = new HttpSolrServer(TEST_SERVER_URL);
3   String hashSum = calculateDigestFor(extractableContent);
4
5   ContentStreamUpdateRequest updateRequest
6       = new ContentStreamUpdateRequest("/update/extract");
7
8   updateRequest.addFile(extractableContent);
9
10  updateRequest.setParam("literal.id",  extractableContent.getName());
11  updateRequest.setParam("fmap.content", "tableofcontents"); // !!!!
12  updateRequest.setParam("ppn", testMapping.get("ppn"));
13  updateRequest.setParam("id",  hashSum);
14  NamedList<Object> response = server.request(updateRequest);
```

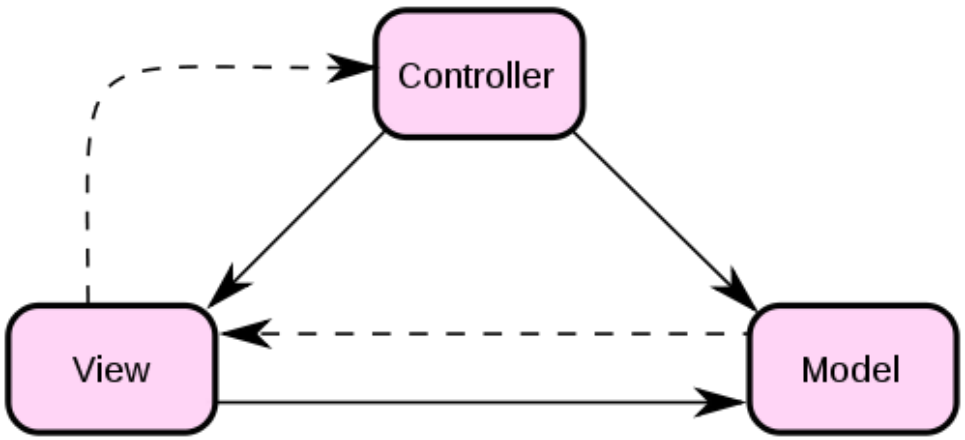
Anforderungen

Anforderungen

- Schlankheit
- Suchergebnis → JSON-Format
  - PPN
  - Index selbst (jeweils für Abstracts, Inhaltsverzeichnisse)
  - ggf. ID (z. B. Digest des ursprünglichen Textes)

MVC

Model-View-Controller Pattern



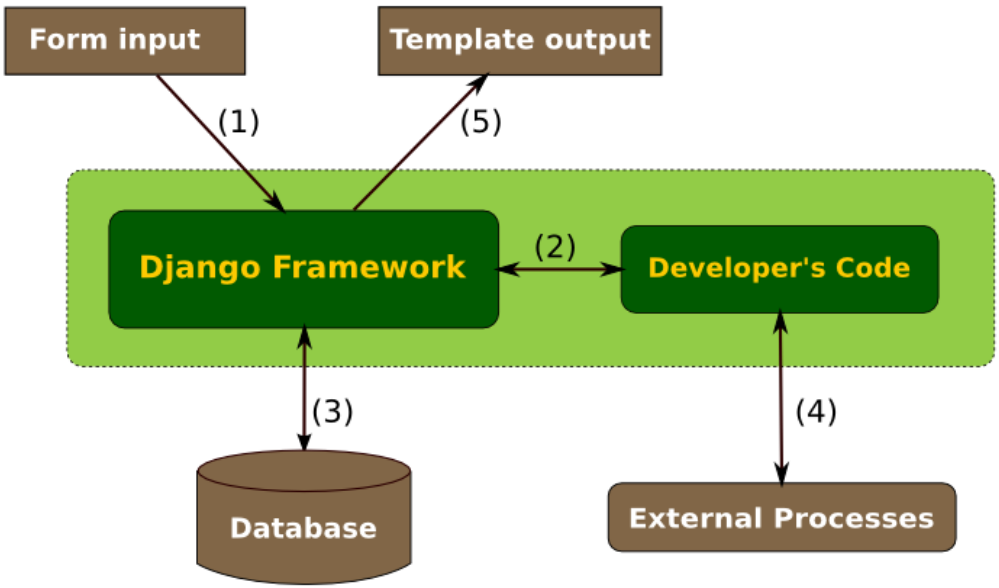
Evaluation

Evaluation

- Solr API
- Frameworks
- Ruby on Rails 3.x vs. Django 1.x
- Ruby 1.9.x vs. Python 2.x
- → vorhandenes Know-How

Django

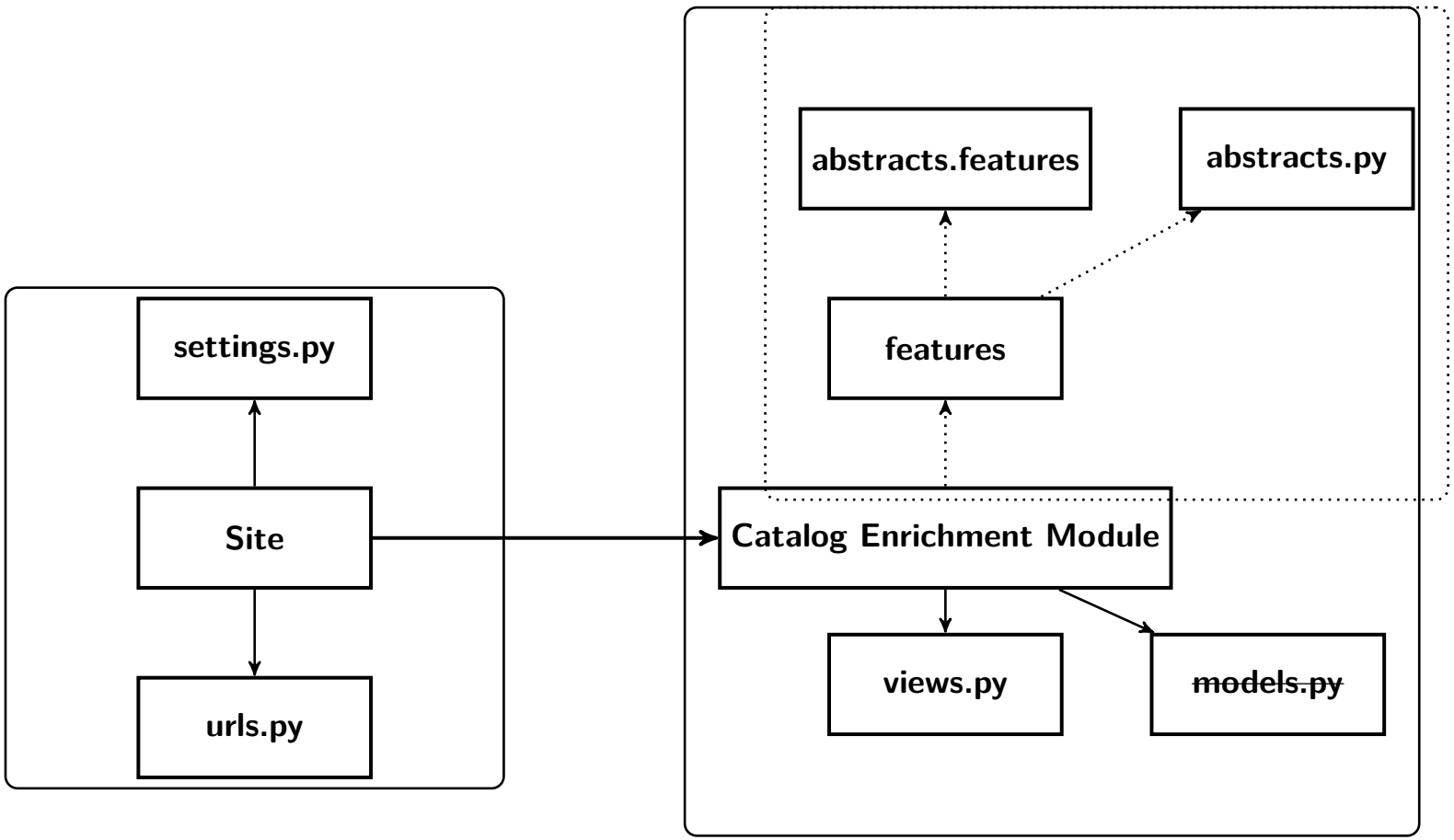
Django Framework



- Python Web framework
- MVC und DRY principle
- Modularisierung & Pluggablity

Django Projekt

Django Projekt Struktur



Python Solr Library

Sunburnt

- vs. *Haystack* → ORM/Models
- ›pythonic‹
- Querying API
- Object-Mapping von Suchergebnissen (Python-Klasse)
- erlaubt die Einschränkung einer Suche auf bestimmte Indexfelder
- Faceting
- Pagination with Django

Content-negotiation & DRY

Rubyish

```ruby
def index
  @people = Person.find(:all)

  respond_to do |format|
    format.html
    format.json { render :json => @people.to_json }
    format.xml { render :xml => @people.to_xml }
  end
end
```

Content-negotiation

Content-negotiation & Django

Pythonic

- Django-Decorators & `MultiResponse` class
- Bennett 2008: *Another take on content negotiation*
- Content-negotiation framework for Django[4]

[4] Oxford-University 2012

Content-negotiation

Pythonic

```
1 class IndexView(JSONView, HTMLView):
2     def get(self, request):
3         # ...
4         response = self.render(request, context, 'index')
5         response['X-Renderer-Format'] = response.renderer.format
6         return response
```

Code

View Snipplet

```
1  class CatalogEnrichment:
2      def __init__(self, id, ppn, abstracts, tableofcontents, **other_kwargs):
3          self.id = id
4          self.ppn = ppn
5          self.abstracts = abstracts
6          self.tableofcontents = tableofcontents
7
8      def __repr__(self):
9          return 'CatalogEnrichment("%s", "%s")' % (id, ppn)
10
11 class QueryAbstractsView(JSONView, HTMLView):
12     def get(self, request):
13         solr = SolrInterface(SOLR_SERVER_URL)
14         result = solr.query(request.GET.get('fulltext')).\
15                  field_limit(["ppn", "abstract", "id"]).\
16                  execute(constructor=CatalogEnrichment)
17         response = self.render(request,\
18                                Context({"solr-result". result}),\
19                                'query-abstracts')
20         response['X-Renderer-Format'] = response.renderer.format
21         return response
```

Django Tests

Django Unit & Interation Tests

- automatisch
- »test harness« → Regression testing, Refactoring
- UnitTests → Mikroebene
- InterationTests → intermeditär
- code-zentrisch
- Fixtures und Testdaten

BDD

Lettuce

- User Stories/Use Cases ↔ Test-Driven Development
- Falcão 2012: »BDD[5] tool for python, 100 % inspired on cucumber.«
- Gherkin language → Features
- Python → Steps (Implementierung)

[5] Behavior Driven Development

SWBplus Ernte

Kataloganreicherung ernten

- kein Webservice
- keine (REST)-API

Lösungsansatz

→ Harvasting von PDF-Dateien

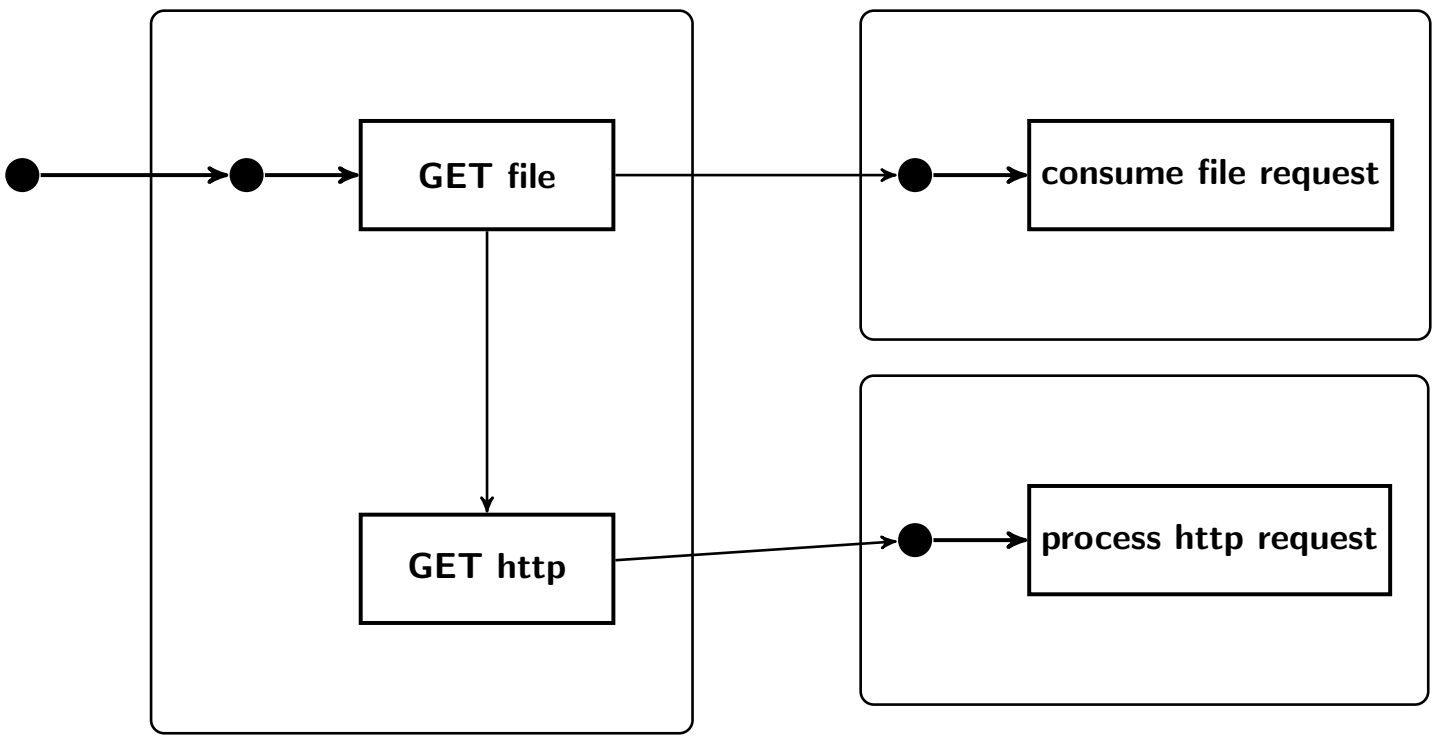
Erntearbeiter

Crawler-Skript

- Datenauszug muss bekannt sein (Liste)
- Crawler-Skript
  - Parallelisierung der HTTP-GET-Requests
  - Metadatenextraktion
    - Aus dem Dateinamen selbst
    - Selektion von HTML-Elementen via XPath/CSS-Selektoren
    - Optional: Fingerabdrücke sammeln (Message-Digest)

Erntearbeiter

Crawler-Skript



Crawler-Skript

Ruby

```ruby
EM.run do
    EM::Synchrony::FiberIterator.new(urls_subset, @concurrency).each do |url|
        url.strip!
        begin
          file_name = URI.parse(url).path
          file_name.gsub!(/^\//, "")
          file_request = EventMachine::HttpRequest.new(url).get
          consume_file_request(file_request, file_name, url, pending)
          unless @ppn_output.empty?
            htm_url = url.gsub!(/\.pdf/, ".htm")
            htm_request = EventMachine::HttpRequest.new(htm_url).get
            consume_http_request(htm_request, file_name, htm_url, pending)
          end
        rescue => e
          handle_error(e)
        end
    end
end
```

Indexierung

Keeper-Skript

- *Solr* `ExtractingRequestHandler` via HTTP-Post-Requests
- Skriptbasiert
  - Python (`httplib`, `twisted.web`)
  - Ruby (`net/http`, `em/http`)
  - Shell (`curl`)
- mittels *Solrj* (Java Application)

Keeper-Skript

Ruby

```ruby
EM::Synchrony::FiberIterator.new(urls_subset, @concurrency).each_key do
|resource_name|
    metadata = @ppn_list[resource_name]
    begin
      type = metadata[resource_name].to_sym
      if @types_selection_map.include_key?(type)
         parameters = build_parameters(resource_name, type, metadata)
         source = File.join(@source_dir, resource_name)
         index_request_uri = "{@solr_uri}{@solr_extract_handler}?#{parameters}"
         index_request =
         EventMachine::HttpRequest.new(index_request_uri).post(:file => source)
         consume_index_request(index_request, source, pending)
     else
        message = "Loremp ipsum"
        @logger.error(message)
     end
    rescue => e
     handle_error(e)
    end
end
```

Reflexion

Kritische Pfade

- Performance und Verhalten unter Last
- Zeichensätze & Encoding (Multi-Language)
- existierender OCR-Layer notwendig (PDF-Dateien)
- Qualität des OCR-Layers
  - ›ii‹ anstelle von ›ü‹ (Umlautproblematik)
  - ›ä‹ anstelle von ›ā‹ (Diakritische Zeichen)
  - Ligaturen (ß)
  - Mixed Languages (Schritzeichen & Alphabetschrift)
  - Worttrennung (Hyphenation)
- ›Test early and fail often.‹

Fazit

Fazit

- Suchindex-Erzeugung auf Basis der Digitalisierungen von SWBplus
- Implementierung eines eigenen (schlanken) Dienstes
- SWPPlus ›Erntemaschine‹
- Digitalisierung → Such-Index

Literatur I

- *Apache Solr* (2012). URL: http://lucene.apache.org/solr/ (besucht am 25. 06. 2012).
- Bennett, James (2008). *Another take on content negotiation*. URL: http://is.gd/gSkIj0 (besucht am 25. 06. 2012).
- *Django Project* (2012). URL: https://www.djangoproject.com/ (besucht am 25. 06. 2012).
- *Event Machine* (2012). URL: http://eventmachine.rubyforge.org/EventMachine.html (besucht am 25. 06. 2012).
- Falcão, Gabriel (2012). *Lettuce*. URL: https://www.djangoproject.com/ (besucht am 25. 06. 2012).
- Gamma, Erich, Richard Helm und Ralph Johnson (2001). *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. München: Addison-Wesley.

Literatur II

- Grigorik, Ilya (2012a). *EM-HTTP-Request*. URL: https://github.com/igrigorik/em-http-request (besucht am 25. 06. 2012).
- – (2012b). *EM-Synchrony*. URL: https://github.com/igrigorik/em-synchrony (besucht am 25. 06. 2012).
- Services, Oxford University Computing (2012). *Content-negotiation framework for Django*. URL: https://github.com/oucs/django-conneg (besucht am 25. 06. 2012).
- *SWBplus* (2012). URL: http://www.bsz-bw.de/digitalebibliothek/swbplus.html (besucht am 25. 06. 2012).